CSE 4600 Operating Systems

Homework #2

All assignments are to be submitted to Canvas. Please note that the due time of homework is at 11:55 pm (Canvas time) on the due date. Please make sure to "submit" after uploading your files. Please do not attach unrelated files. You will not be able to change your files after clicking submit.

When you log onto the Linux lab JB359 computers, the operating system starts running a *program* called the *shell*. The standard shell for the Linux operating system running on Linux lab JB359 computers is *Bash*. The purpose of the shell is to package system services under the control of an *interactive command interpreter* whose external interface is suitable for human consumption. When the shell program (a *passive* entity) is run, it becomes a *process* (an *active* entity). The shell (process) outputs a *command prompt* to the screen and then waits to read input entered from the keyboard (i.e., the shell is actually temporarily suspended until you type some characters). After you type some characters and press the return key to end the line, the shell re-starts and reads these characters into an internal array. The shell then analyzes the contents of the internal array to extract *commands* and *arguments*. For some commands, such as `cd` (i.e., change the current directory), the shell can execute the command directly (i.e., the command is *built-in* to the shell). When the shell finishes executing a built-in command, it outputs another command prompt to the screen. For other commands, such as `ls` (i.e., list the contents of the specified directory), the shell searches directories specified in the shell `path` environment variable, looking for a file named "ls" (type `env` at the command prompt to see the contents of the `path` environment variable and other environment variables). If the file is found and the file is executable, the shell will invoke a service of the operating system to run the file, thereby creating a new process. While the shell is waiting for the new process to finish executing, it is temporarily suspended. When the new process finishes executing, the shell re-starts and outputs another command prompt to the screen.

A shell program is not really special in any way. In fact, there are actually many different shells available to choose from. And if you don't like any of the available shells, you could write your own given the opportunity and sufficient motivation (e.g., see below). Read some Bash documentation to familiarize yourself with what a shell is, the features a shell provides, and the types of operations a shell support. Some useful documentation can be found by typing `man bash` at the command prompt.

In this homework, you are required to write a shell program, called `myshell`, that runs on the Linux operating system (specifically, one of the Linux machines in JB359). You can log onto these machines from your Coyote account.

PART 1 Implementing myshell and Capturing Result 40 points

Your shell program must provide services to support the following commands and features:

Note: The commands are shown in uppercase font, but don't have to be programmed that way. The parameters to the commands are enclosed between the symbols < and >, but the symbols are not actually part of the parameters.

a. **(3 marks)** `STOP`: Terminates execution of the current `myshell` session.

b. **(3 marks)** `SETSHELLNAME` <*shell_name*>: Sets the shell name in the `myshell` command prompt to <*shell_name*>. For example, a sample command prompt is shown below:

   ```
   myshell>
   ```

   In this command prompt, there are two parts. The first part, `myshell`, is the shell name. The second part, >, is the terminator. If no shell name is defined, `myshell` should be the default shell name.

c. **(3 marks)** `SETTERMINATOR` <*terminator*>: Sets the terminator in the `myshell` command prompt to <*terminator*>. If no terminator is defined, `myshell` should use > as the default terminator.

d. **(10 marks)** `NEWNAME` <*new_name*> | <*new_name*> <*old_name*>: Manages the alias list. The first option deletes a previously defined alias. The second option defines an alias for another command. For example, the command `NEWNAME mymove` deletes the alias for `mymove`, the command `NEWNAME mycopy cp` defines `mycopy` as the alias for the `cp` command. If an alias for a command already exists, then the new alias replaces the old alias. The maximum number of aliases in the alias list should be set to 10 as the default.

e. **(6 marks)** `LISTNEWNAMES`: Outputs all the aliases that have been defined. Each pair of names should be shown on one line. For example, the possible aliases for a few commands are shown below:

   ```
   mycd cd
   mycopy cp
   ```

f. **(6 marks)** `SAVENEWNAMES` <*file_name*>: Stores all currently defined aliases in the file <*file_name*>.

g. **(5 marks)** `READNEWNAMES` <*file_name*>: Reads all aliases in the file <*file_name*>.

h. **(2 marks)** <*UNIX_command*>: Executes the UNIX command <*UNIX_command*>, corresponding to any valid UNIX command. If the first token on a command line is not a built-in command, assume that it is a UNIX command.

i. **(2 marks)** HELP: shows all built-in commands supported by myshell.

Note: You must handle all the built-in commands with exactly the same syntax as shown above. Thus, an important part of the `myshell` program will be to parse commands entered at the command prompt to break them down into their component parts. Once a command has been parsed, the component parts can be checked to ensure that a valid command has been entered and that it adheres to the required syntax.

For this problem, the results are worth 40 marks out of the 50 marks available. The breakdown of these marks is shown above. Demonstrate that your shell works and that it can handle all of the requirements described above. Your demonstration should be captured in a script log file and follow the sequence of commands given below:

```
HELP
ls –l
SETSHELLNAME mysh
SETTERMINATOR $
NEWNAME myhelp help
NEWNAME mycopy cp
NEWNAME mycat cat
NEWNAME mydel rm
SAVENEWNAMES myaliases.txt
LISTNEWNAMES
mycat myaliases.txt
mycopy myaliases.txt myfile
mycat myfile
cat myfile
ls
LISTNEWNAMES
READNEWNAMES myaliases.txt
STOP
```

PART 2 Error Handling 5 marks

Finally, demonstrate that your shell can handle errors. Your shell program should effectively identify and recover from errors. For example, bad input and/or the inability to execute a command should not cause `myshell` to crash. This will be worth 5 marks out of the 50 marks available. In order to receive all 5 marks, you must be able to handle at least 5 different kinds of errors. Your demonstration should be captured in a script log file.

PART 3 Programming Style 5 marks

Please use the following style when programming:
a. Opening Comment block: there should be comments at the beginning of the program describing

- Name, ID, the date you write this program.
- Purpose: what the program does.

b. Comment block for member functions: there should be comments before the member function definition describing

- Purpose: explain what the function does.
- Parameter(s): explain the purpose of each parameter to the function.

- Return: if this function sends back a value via a return statement, describe the purpose of that value here.

c. Indentation/ spacing:

- Indenting each subcomponent or statements two or four spaces more than the construct within which it is nested.
- A single space should be added on both sides of a binary operator.
- Place braces (i.e., { and }) on separate lines, aligned with one another and the preceding line.

d. The descriptive identifiers make programs easy to read.

Deliverables:

You can work alone or in a partnership.
If you are working in a partnership, only one of members of your group submits the assignment to Canvas on behalf of your group. Please add the team members' names and IDs at the opening comment block of source code file hw02.c/cpp.

Please submit to Canvas:
(1) your source code files *hw02.c/cpp* (i.e., only the `.cpp/c` and `.h` files),
(2) a single script log file *hw02log.txt* showing the correct execution of your shell, and
(3) a single script log file *hw02errorlog.txt* showing the error handling capabilities of your shell.